**Update Customer Process (Step-by-Step)**

**Update Customer (Step-by-Step)**:

1. Update list-customers.jsp
   1. New “Update” link
2. Create customer-form.jsp
   1. Prepopulate the form
3. Process form data
   1. Controller > Service > DAO

**1) Update list-customer.jsp**:

Add code to the list-customer.jsp pade

<table>

<tr>

<th>First Name</th>

<th>Last Name</th>

<th>Email</th>

<th>Action</th>

</tr>

<!-- Loop over and print out customer -->

<c:forEach var=*"tempCustomer"* items=*"*${customers}*"*>

<!-- construct an "update" link with customer id -->

<c:url var=*"updateLink"* value=*"/customer/showFormForUpdate"*>

<c:param name=*"customeId"* value=*"*${tempCustomer.id}*"* />

</c:url>

<tr>

<td>${tempCustomer.firstName}</td>

<td>${tempCustomer.lastName}</td>

<td>${tempCustomer.email}</td>

<td>

<a href=*"*${updateLink}*"*>Update</a>

</td>

</tr>

</c:forEach>

</table>

Now under Action Column Update link will be created.

The update link contains the customer id as follow.

<http://localhost:8080/spring-mvc-crud-demo/customer/showFormForUpdate?customeId=1>

<http://localhost:8080/spring-mvc-crud-demo/customer/showFormForUpdate?customeId=2>

**2) Create customer-form.jsp**:

Now we have to write the code for "*showFormForUpdate*" in our controller.

<c:url var=*"updateLink"* value=*"/customer/showFormForUpdate"*>

**Update CustomerController.java**:

// mapping for "/customer/showFormForUpdate"

@GetMapping("/showFormForUpdate")

**public** String showFormForUpdate(@RequestParam("customeId") **int** theId, Model theModel) {

// get the customer from the database

Customer theCustomer = customerService.getCustomer(theId);

// set customer as a model attribute to pre-populate the form

theModel.addAttribute("customer", theCustomer);

//send over to our form

**return** "customer-form";

}

Now when user click the update link, it will go to the “**customer-form.jsp**” page and the form field will fill up with the related customer value ("**firstName**", "**lastName**", and "**email**"). For this we have to update our update our **“service”** and “**DAO**”.

**Service**:

Add method

**CustomerService.jave**:

**public** **interface** CustomerService {

...

**public** Customer getCustomer(**int** theId);

}

**CustomerServiceImpl.java**:

@Service

**public** **class** CustomerServiceImpl **implements** CustomerService {

// need to inject CustomerDAO

@Autowired

**private** CustomerDAO customerDAO;

...

@Override

@Transactional

**public** Customer getCustomer(**int** theId) {

**return** customerDAO.getCustomer(theId);

}

}

**DAO**:

**CustomerDAO.java**:

**public** **interface** CustomerDAO {

...

**public** Customer getCustomer(**int** theId);

}

**CustomerDAOImp.java**:

@Repository

**public** **class** CustomerDAOImp **implements** CustomerDAO {

// need to inject the session factory

@Autowired

SessionFactory sessionFactory;

...

@Override

**public** Customer getCustomer(**int** theId) {

// get the current Hibernate session

Session currentSession = sessionFactory.getCurrentSession();

// now retrieve/read from database using the primary key

Customer theCustomer = currentSession.get(Customer.**class**, theId);

**return** theCustomer;

}

}

In our CustomerController we add code for show the form for update. In the background we actually get the customer from our service, we send it as a model attribute and then we send it over to our customer form.

Here the key think here is the model attribute. That’s the same attribute name that the form will use when they pre-populate the data.

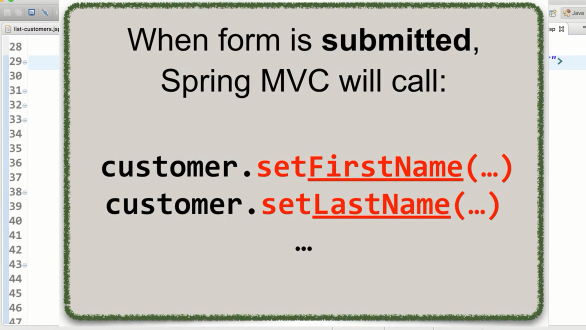


Again, whatever data that we have populate for that attribute, they are going to use that each one of those fields here for "**firstName**", "**lastName**", and "**email**".

Behand the scenes, Spring will call the appropriate getter methods when the form is loaded.



And when we submit the data Spring actually call the setter method.

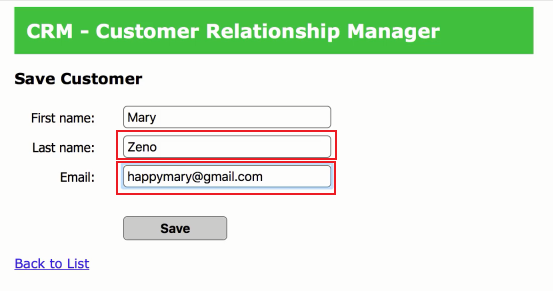


So,

**Form Load: Call getter**

**Form Submit: Call getter**

Now after run the project if we click the “**Update**” link we will see the “**customer-form.jsp**” page as follow.



**3) Process form data**:

Now for process form data we have to update **"Controller > Service > DAO"**

**Add extra information in "customer-form.jsp"**:

Before process data (Save update data into database) we have to add some additional information into **"customer-form.jsp"** page. We have to associated the data with customer id. We can accomplish that by adding a hidden form field and provide that customer id.

<form:form action=*"saveCustomer"* modelAttribute=*"customer"* method=*"POST"*>

<!-- need to associate this data with customer id -->

<form:hidden path=*"id"* />

When the form is loaded spring-MVC will call "**customer.getId()**" and when the form is submit spring-MVC will call "**customer.setId()**" with the appropriate data.

This line is very important. Without this line we actually lose context or lose the id of the original customer. The existing value not updates in the database it will save as a new value.

**Update CustomerDAOImp.java**:

Now for update the value we don’t change anything in our CustomerController class. We update the **saveCustomer()** method in **CustomerDAOImp.java** class. We replace the save() method with method saveOrUpdate()

@Override

**public** **void** saveCustomer(Customer theCustomer) {

// get current Hibernate session

Session currentSession = sessionFactory.getCurrentSession();

// save the customer ...

currentSession.saveOrUpdate(theCustomer);

}

In the Hibernate API, there are two ways to adding data to the database.



But in this case, we actually using the same method here for adding and for updating. We will use other method called **saveOrUpdate().**



Now we have same method for insert and update.

Update Customer Process (Step-by-Step)